APPENDIX II

```
/*
 $Id: galois.c,v 1.4 2000/03/21 14:49:27 rollins Exp $

 galois.c

 (c) 2000 Chameleon Systems Inc.
     Algorithms in Reconfigurable Silicon

 Mark Rollins
 14-Mar-2000
*/

#include "galois.h"

/*
 -----------------------------------------------------------
 Polynomial Multiplication Modulo f(x) for GF(2^25)

 This version is hard coded for GS(2^25).

 Since we need to multiply polynomials of order 24, the result
 will not fit in a 32-bit register. We need to manage two
 such registers (upper and lower).

 Polynomial multiplication is just simply shifts and adds
 with the added complexity of managing the two registers.

 Polynomial reduction modulo f(x) is performed by adding in
 shifted correction terms f(x) which line up with the undesired
 higher order 1's in the product (lying in the 25-th bit and above).
 -----------------------------------------------------------
*/

int poly_mult_modulo_fx_2p25( int a_x, int b_x, int f_x )
{
    int lower, upper;
    int shft_a, shft_b, upper_b;
    int fsl, fsu;
    int i;

    /* Initialize */
    upper = 0;
    upper_b = 0;
    shft_a = a_x;
    shft_b = b_x;
    lower = ((shft_a & 1) == 1) ? b_x : 0; /* Shift 0 */

    /* Shifts 1 to 7 remain in lower register */
    for (i=1; i <= 7; i++) {
      shft_a >>= 1;
      shft_b <<= 1;
      if ((shft_a & 1) == 1)
          lower = lower ^ shft_b;
    }

    /* Shifts 8 to 24 spread across lower & upper registers */
    for (i=8; i <= 24; i++) {
```

```
        shft_a >>= 1;
        upper_b <<= 1;
        upper_b += ( (shft_b&0x80000000) >> 31 );
        shft_b <<= 1;
        if ((shft_a & 1) == 1) {
            lower = lower ^ shft_b;
            upper = upper ^ upper_b;
        }
    }

    /*
       Perform modulo f(x) reduction on high order bits:
       - check if bits 48, 47, ..., 25 are unity
       - if yes, add shifted versions of f(x)
    */

    /* Bits 48 to 32 spread across lower & upper registers */
    fsu = f_x >> 9;
    for (i=16; i>=0; i--) {
      fsl = f_x << (7+i);
      if ( ((upper>>i)&1) == 1 ) {
          upper = upper ^ fsu;
          lower = lower ^ fsl;
      }
      fsu >>= 1;
    }

    /* Bits 31 to 25 remain in lower register */
    for (i=31; i >= 25; i--) {
      fsl = f_x << (i-25);
      if ( ((lower>>i)&1) == 1 )
          lower = lower ^ fsl;
    }
/*     printf("Upper: %x\n", upper); */
/*     printf("Lower: %x\n", lower); */

    return lower;
}

/*
 --------------------------------------------------------------
 Polynomial Division With Max Degree 25

 Determine a(x) = ( g(x) / f(x) )_{deg<25}

 where
           f(x) is a primitive polynomial of degree '25'
         g(x) is a polynomial of degree < '25'
         a(x) is a polynomial of degree < '25'

 The higher order terms of a(x) are not calculated.
 --------------------------------------------------------------
*/

int poly_divide_max_degree_25( int g_x, int f_x )
{
    int i, a_x, shft_g;
```

```
    a_x = 0;
    shft_g = g_x;
    for (i=0; i < 25; i++) {
      if ( (shft_g & 1) == 1 ) {
          shft_g = shft_g ^ f_x;
          a_x += (1 << i);
      }
      shft_g >>= 1;
    }
    return a_x;
}

/*
 -----------------------------------------------------------
 Polynomial Multiplication With Max Degree 25
 Determine
            g(x) = ( f(x)b(x) )_{deg<25}
 where
            f(x) is a primitive polynomial of degree '25'
            b(x) is a polynomial of degree < '25'
            g(x) is restricted to have degree < '25'

 The higher order terms of g(x) are not calculated.

 The primitive polynomial f(x) is specified by its primitive polynomial.

 The lower order polynomial terms are stored in LSB's.
 -----------------------------------------------------------
*/

int poly_mult_max_degree_25( int a_x, int f_x )
{
    int i, g_x, shft_f;

    g_x = 0;
    shft_f = f_x;
    for (i=0; i <= 25; i++) {
      if ( (shft_f & 1) == 1 )
          g_x = g_x ^ ( a_x << i );
      shft_f >>= 1;
    }
    g_x = g_x & 0x1FFFFFF;    /* Keep bits 0 through 24 */
    return g_x;
}

/*
 -----------------------------------------------------------
 Polynomial Multiplication With Max Degree 25 for UMTS Top Polynomial
 Determine
            g(x) = ( f(x)b(x) )_{deg<25}
 where
            f(x) is a 1 + x^3 + x^25
            b(x) is a polynomial of degree < '25'
            g(x) is restricted to have degree < '25'

 The higher order terms of g(x) are not calculated.
```

```
 The primitive polynomial f(x) is specified in the UMTS Standard.

 The lower order polynomial terms are stored in LSB's.
 -------------------------------------------------------------
*/

int poly_mult_max_degree_UMTS_top( int a_x )
{
    int g_x;

    g_x = a_x;
    g_x = g_x ^ ( a_x << 3 );
    g_x = g_x ^ ( a_x << 25 );
    g_x = g_x & 0x1FFFFFF;    /* Keep bits 0 through 24 */
    return g_x;
}

/*
 -------------------------------------------------------------
 Polynomial Multiplication With Max Degree 25 for UMTS Bottom Polynomial
 Determine
           g(x) = ( f(x)b(x) )_{deg<25}
 where
           f(x) is a 1 + x + x^2 + x^3 + x^25
           b(x) is a polynomial of degree < '25'
           g(x) is restricted to have degree < '25'

 The higher order terms of g(x) are not calculated.

 The primitive polynomial f(x) is specified in the UMTS Standard.

 The lower order polynomial terms are stored in LSB's.
 -------------------------------------------------------------
*/

int poly_mult_max_degree_UMTS_bot( int a_x )
{
    int g_x;

    g_x = a_x;
    g_x = g_x ^ ( a_x << 1 );
    g_x = g_x ^ ( a_x << 2 );
    g_x = g_x ^ ( a_x << 3 );
    g_x = g_x ^ ( a_x << 25 );
    g_x = g_x & 0x1FFFFFF;    /* Keep bits 0 through 24 */
    return g_x;
}

/* -------------------------------------------------------------
   Bit Reverse a 25-bit Integer
   ------------------------------------------------------------- */

int bit_reverse_25( int g_x )
{
    int i, r_x;
    r_x = 0;
```

```
    for(i=0; i < 25; i++) {
      r_x <<= 1;
      r_x += (g_x>>i) & 1;
    }
    return r_x;
}

/*
 ---------------------------------------------------------------
 LFSR Generator for N=25 with Mask Polynomial
 ---------------------------------------------------------------
*/

#ifndef ARC
int LFSR_gen_25_mask( int f_x, int *a_x, int m_x )
{
    int i, lsb, rxor, new_msb;

    /* Calculate LSB using mask */
    rxor = m_x & *a_x;
    lsb = 0;
    for (i=0; i <= 24; i++) {
      lsb = lsb ^ (rxor & 1);
      rxor >>= 1;
    }

    /* Calculate NEW MSB */
    rxor = f_x & *a_x;
    new_msb = 0;
    for (i=0; i <= 24; i++) {
      new_msb = new_msb ^ (rxor & 1);
      rxor >>= 1;
    }

    /* Update state */
    *a_x = (*a_x>>1) ^ ( new_msb << 24 );

    return lsb;
}
#endif

/*
 ---------------------------------------------------------------
 LFSR Generator for N=25
 ---------------------------------------------------------------
*/

#ifndef ARC
int LFSR_gen_25( int f_x, int *a_x )
{
    int i, lsb, rxor, new_msb;

    /* Extract LSB */
    lsb = *a_x & 1;

    /* Calculate NEW MSB */
    rxor = f_x & *a_x;
```

```
    new_msb = 0;
    for (i=0; i <= 24; i++) {
      new_msb = new_msb ^ (rxor & 1);
      rxor >>= 1;
    }

    /* Update state */
    *a_x = (*a_x>>1) ^ ( new_msb << 24 );

    return lsb;
}
#endif

/*
 -------------------------------------------------------------
 Print Bitstring
 where 'n' is the number of bits, 1 < n <= 32
 -------------------------------------------------------------
*/

#ifndef ARC
#include <stdio.h>

void print_bitstring( char *mesg, int poly, int n )
{
    int i;
    for(i=n-1; i >= 0; i--)
      printf("%1d ", (poly>>i) & 1);
    printf(mesg);
    printf("\n");
}
#endif

/*
 -------------------------------------------------------------
 Reduction of x^power Modulo f(x) to a polynomial
 where
         f(x) = 1 + x^3 + x^25
 -------------------------------------------------------------
*/

#ifndef ARC
#include <math.h>

int reduce_25( int power )
{
    int index, result, stop;
    short int *list = (short int*) calloc( power+1, sizeof(short int) );

    for (index=power; index >= 0; index--)
      list[index] = 0;

    list[power] = 1;
    index = power;

    while (index >= 25) {
```

```
        if (list[index]) {
            list[index] = 0;
            list[index-22] = list[index-22]^1;
            list[index-25] = list[index-25]^1;
        }
        index -= 1;
    }
    result = 0;
    stop = ( power < 25 ) ? power+1 : 25;
    for (index=0; index < stop; index++)
        result += (list[index] << index);
    free(list);
    return result;
}
#endif

/*
 ------------------------------------------------------------
 Given a polynomial g(x), calculate the value of 'k' in

           g(x) = x^k modulo f(x)
 ------------------------------------------------------------
*/

#ifndef ARC
int revert_modulo_poly_reduction( int g_x, int f_x )
{
    int cnt = 0;
    int shft = g_x;
    while (shft != 1) {
        if ( (shft & 1) == 0 ) {
            do {
                shft >>= 1;
                cnt++;
            } while ( (shft & 1) == 0 );
        }
        else
            shft = shft ^ f_x;
    }
    return cnt;
}
#endif

/*
 ------------------------------------------------------------
 Print a polynomial as a sum of powers of 'x'
 ------------------------------------------------------------
*/

#ifndef ARC
void print_poly_25( int g_x )
{
    int i, bit;
    for(i=0; i < 25; i++) {
        bit = (g_x>>i)&1;
        if (bit) {
            if (i==0)
```

```
            printf("1");
          else
            printf(" + x^%d",i);
        }
      }
    printf("\n");
}
#endif
```

```
/*
 $Id: galois.h,v 1.4 2000/03/21 14:49:36 rollins Exp $

 galois.h

 (c) 2000 Chameleon Systems Inc.
    Algorithms in Reconfigurable Silicon

 Mark Rollins
 14-Mar-2000
*/

#ifndef _galios_h_

/* -----------------------------------------------------------
   ARC Routines:
   ----------------------------------------------------------- */

int bit_reverse_25(int g_x);

int poly_mult_max_degree_UMTS_top( int a_x );

int poly_mult_max_degree_UMTS_bot( int a_x );

int poly_mult_modulo_fx_2p25( int a_x, int b_x, int f_x );

int poly_divide_max_degree_25( int g_x, int f_x );

int poly_mult_max_degree_25( int a_x, int f_x );


/* -----------------------------------------------------------
   Solaris/Debugging Routines:
   ----------------------------------------------------------- */
#ifndef ARC

int LFSR_gen_25_mask( int f_x, int *a_x, int m_x );
```

```
int LFSR_gen_25( int f_x, int *a_x );

void print_bitstring( char *mesg, int poly, int n );

int revert_modulo_poly_reduction( int g_x, int f_x );

void print_poly_25( int g_x );

int reduce_25( int power );

#endif

#define _galois_h_ 1
#endif
```

```c
/*
 $Id: galois_arc.c,v 1.3 2000/03/21 00:19:27 rollins Exp $

 galois_arc.c

 (c) 2000 Chameleon Systems Inc.
     Algorithms in Reconfigurable Silicon

 Mark Rollins
 14-Mar-2000
*/

#include "galois.h"

#define N_bits 1000

#define mask  0x0040090
#define poly1 0x2000009
#define user  0x1000000

int main( int argc, char **argv )
{
    int a1x, a1x_rev, a2x, a2x_rev;
    int g1x, g2x;

    a1x_rev = user;

    /* Determine new seed required to produce a delayed
       version of the LFSR sequence
    */
    a1x = bit_reverse_25( a1x_rev );
    g1x = poly_mult_max_degree_UMTS_top( a1x );
    g2x = poly_mult_modulo_fx_2p25( g1x, mask, poly1 );
    a2x = poly_divide_max_degree_25( g2x, poly1 );
    a2x_rev = bit_reverse_25( a2x );
}
```

```
/*
 $Id: galois.h,v 1.4 2000/03/21 14:49:36 rollins Exp $

 galois.h

 (c) 2000 Chameleon Systems Inc.
     Algorithms in Reconfigurable Silicon

 Mark Rollins
 14-Mar-2000
*/

#ifndef _galios_h_

/* ---------------------------------------------------------------
   ARC Routines:
   --------------------------------------------------------------- */

int bit_reverse_25(int g_x);

int poly_mult_max_degree_UMTS_top( int a_x );

int poly_mult_max_degree_UMTS_bot( int a_x );

int poly_mult_modulo_fx_2p25( int a_x, int b_x, int f_x );

int poly_divide_max_degree_25( int g_x, int f_x );

int poly_mult_max_degree_25( int a_x, int f_x );


/* ---------------------------------------------------------------
   Solaris/Debugging Routines:
   --------------------------------------------------------------- */
#ifndef ARC

int LFSR_gen_25_mask( int f_x, int *a_x, int m_x );

int LFSR_gen_25( int f_x, int *a_x );

void print_bitstring( char *mesg, int poly, int n );

int revert_modulo_poly_reduction( int g_x, int f_x );

void print_poly_25( int g_x );

int reduce_25( int power );

#endif

#define _galois_h_ 1
#endif
```

```c
/*
 $Id: galois_tst.c,v 1.2 2000/03/21 00:17:59 rollins Exp $

 galois_tst.c

 (c) 2000 Chameleon Systems Inc.
     Algorithms in Reconfigurable Silicon

 Mark Rollins
 20-Mar-2000
*/

#include "galois.h"

#define N_bits 1000

#define mask  0x0040090
#define poly1 0x2000009

int main( int argc, char **argv )
{
    int a1x, a1x_rev, a2x, a2x_rev;
    int g1x, g2x;
    int seed_ref, seed_del, seed_msk;
    int bits_ref[N_bits], bits_del[N_bits], bits_msk[N_bits];
    int errnum;
    int i,j;

    for (i=0; i <= 0xFFFFFF; i++) {

      a1x_rev = 0x1000000 + i;

      a1x = bit_reverse_25( a1x_rev );
      g1x = poly_mult_max_degree_UMTS_top( a1x );
      g2x = poly_mult_modulo_fx_2p25( g1x, mask, poly1 );
      a2x = poly_divide_max_degree_25( g2x, poly1 );
      a2x_rev = bit_reverse_25( a2x );

      seed_ref = a1x_rev;
      seed_del = a2x_rev;
      seed_msk = a1x_rev;

      errnum = 0;

      for (j=0; j < N_bits; j++) {
          bits_ref[j] = LFSR_gen_25( poly1, &seed_ref );
          bits_msk[j] = LFSR_gen_25_mask( poly1, &seed_msk, mask );
          bits_del[j] = LFSR_gen_25( poly1, &seed_del );
          errnum += ( bits_msk[j] ^ bits_del[j] );
      }

      printf("Undelayed Reference Bits\n");
      for (j=0; j < N_bits; j++)
          printf("%1d", bits_ref[j]);
      printf("\n");

      printf("Delayed Bits - Obtained with Mask\n");
```

```
        for (j=0; j < N_bits; j++)
            printf("%1d", bits_msk[j]);
        printf("\n");

        printf("Delayed Bits - Obtained with Seed\n");
        for (j=0; j < N_bits; j++)
            printf("%1d", bits_del[j]);
        printf("\n");

        printf("Number of errors: %d\n", errnum );
    };
}
```